**Web-scraping avec Python**

Imaginez que vous voulez collecter toutes les annonces de location des appartements à Paris afin de comparer leur prix en fonction de la superficie, l’arrondissement, l’agence, le nombre de pièces et de chambres. Une façon de faire consiste à se rendre sur le site seloger.com et copier manuellement page par page les détails de chaque annonce. Ce travail vous prendra des heures, voire des jours, car il y a plus de 5000 annonces à parcourir !

Commençons par créer une liste des pages web à consulter. Sur le site, les résultats de recherche d’appartements à Paris s’affichent sur environ 295 pages. Chaque page regroupe 20 annonces. Voici les adresses des trois premières pages :

[https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=**1**](https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=1)

[https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=**2**](https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=2)

[https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=**3**](https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=3)

Vous avez remarqué qu’elles sont quasiment identiques. Afin d’obtenir une liste des 295 pages web à consulter, il suffit juste de modifier **le dernier chiffre** indiquant le numéro de la page. On peut le faire avec la fonction suivante :

token = 'https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg='def get\_pages(token, nb):  
 pages = []  
 for i in range(1,nb+1):  
 j = token + str(i)  
 pages.append(j)  
 return pagespages = get\_pages(token,295)

Navigation dans le document HTML

Nous avons une liste des pages web. Maintenant, comment est-ce qu’on accède à chaque page ?

En Python c’est très simple. Trois lignes de code suffisent :

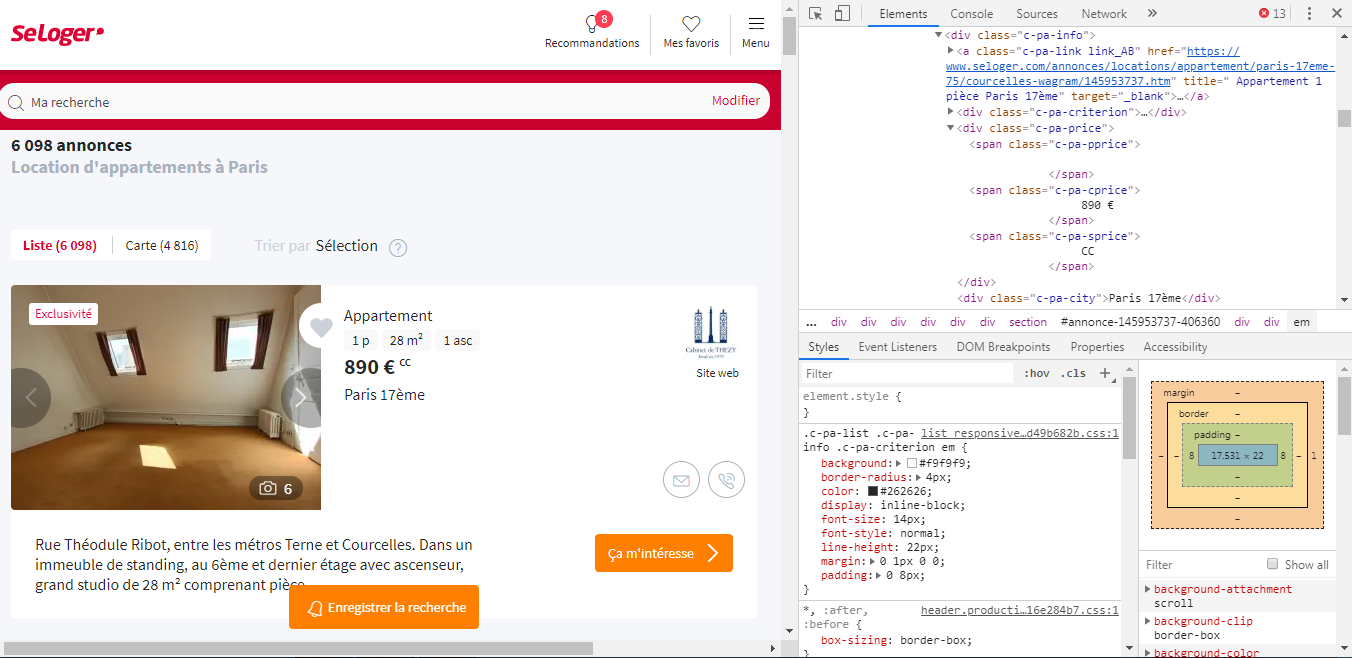
import requests

for i in pages:  
 response = requests.get(i)

A chaque itération la fonction requests.get() essaie de se connecter à une adresse dans notre liste et enregistre le résultat dans la variable response. Si la tentative est réussie, la variable response contiendra la valeur <Response [200]> ainsi que le code source de la page à laquelle on souhaite accéder. Utilisez response.text pour l’afficher.

Les pages web sont écrites en HTML. Le code HTML contient des balises qui aident les navigateurs à afficher correctement le contenu de la page.

Le code HTML du site seloger.com est beaucoup plus complexe. On peut le visualiser à l’aide du navigateur Chrome en faisant une clique droite sur la page affichée et en choisissant “Inspecter”. Chrome est doté d’une fonctionnalité qui permet la recherche des balises dans le code de la page à partir de mots clés.



Il s’avère que les informations que nous voulons extraire se trouvent entre les balises <em>...</em> :

<em class="agency-website" data-codeinsee="750118" data-codepostal="75018" data-idagence="46600" data-idannonce="119026673" data-idtiers="22739" data-idtypepublicationsourcecouplage="SL" data-nb\_chambres="0" data-nb\_photos="6" data-nb\_pieces="1" data-position="3" data-prix="999 €" data-produitsvisibilite="AD:AC:BB:BX:AW" data-surface="32,2400016784668" data-typebien="1" data-typedetransaction="1"> Site web </em>

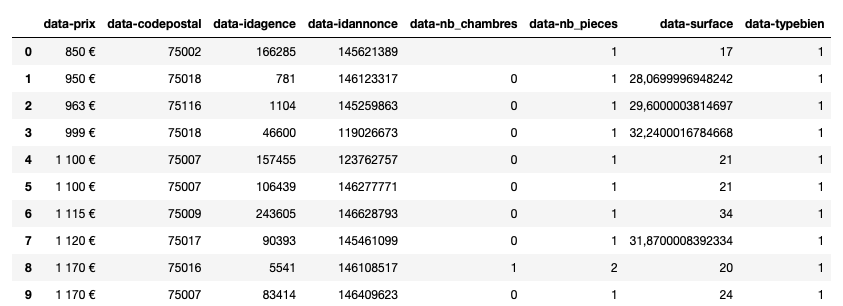
Pour accéder aux informations de cette balise, il faudra la retrouver dans le code HTML de notre variable response. On peut le faire à l’aide de la librairie **BeautifulSoup**. Nous transformons d’abord response en objet BeautifulSoup et cherchons ensuite toutes les balises <em>...</em> à l’aide de la fonction find\_all(). On précise entre parenthèses que nous voulons les balises <em>...</em> qui portent le nom class="agency-website" :

import bs4soup = bs4.BeautifulSoup(response.text, 'html.parser')  
em\_box = soup.find\_all("em", {"class":"agency-website"})

Il nous reste à extraire une par une les informations qui nous intéressent. La variable em\_box fonctionne comme un dictionnaire. Quand on donne une clé à un de ses éléments, par exemple, em\_box[3]['data-prix'], elle nous renvoie sa valeur: '999 €'. Nous pouvons donc faire une boucle qui extrait toutes les valeurs qu’on désire et les range dans une table :

import pandas as pdparameters = ['data-prix','data-codepostal','data-idagence','data-idannonce','data-nb\_chambres','data-nb\_pieces','data-surface','data-typebien']df\_f = pd.DataFrame()  
for par in parameters:  
 l = []  
 for el in em\_box:  
 j = el[par]  
 l.append(j)  
 l = pd.DataFrame(l, columns = [par])  
 df\_f = pd.concat([df\_f,l], axis = 1)

Les deux dernières lignes de code regroupent les valeurs extraites en colonnes et joignent ces colonnes ensemble. Et voilà, nous avons réussi à scraper une page :



Maintenant il faut répéter la même opération pour les 294 pages qui restent. Cela paraît simple au premier abord, mais au bout de quelques itérations votre fonction requests.get()vous enverra la variable response suivante :

<!DOCTYPE html>  
  
<html>  
<head>  
<meta content="noindex,nofollow" name="robots"/>

Les informations contenues dans la balise <meta/> suggèrent que seloger.com vous a identifié comme un **robot** et refuse de vous envoyer la page demandée.

En général, les sites se protègent contre les robots, parce que ces derniers perturbent leur fonctionnement. On distingue deux types de robots : les bons et les mauvais.

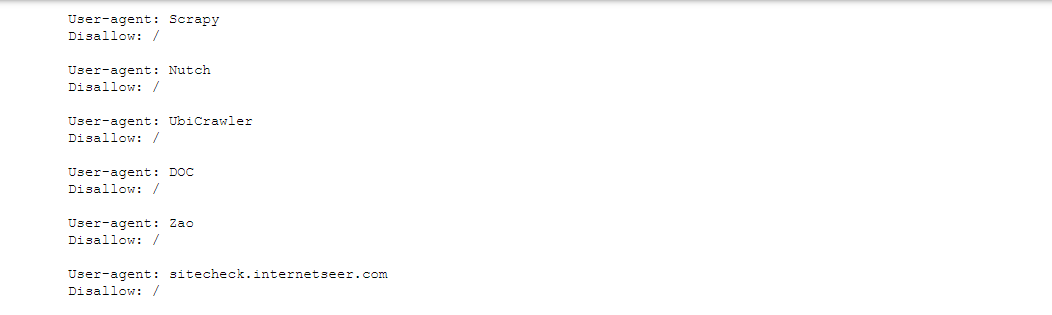
Les bons robots aident les moteurs de recherches à trouver votre site, surveillent sa santé et récupèrent des flux RSS.

Les mauvais robots essayent de planter votre site, télécharger son contenu ou d’envoyer des spams.

Pour connaître quels robots sont autorisés à se rendre sur un site, il faut lire son fichier robots.txt. Avec Python, on peut le faire de la manière suivante :

from urllib.request import urlopenwith urlopen("<https://www.seloger.com/robots.txt>") as stream:  
 print(stream.read().decode("utf-8"))

Voici un extrait de robots.txt du site seloger.com :



On constate qu’il bannit les robots comme Srapy, Zao, UbiCrawler, Nutch. Le nom du robot est précédé par User\_agent. C’est une sorte de carte d’identité de l’utilisateur collectée par les sites.

Les robots se présentent simplement avec leur noms. Cependant, si on accède à seloger.com depuis un navigateur, le champ User\_agent contiendra le nom de notre navigateur et les informations concernant notre système d’exploitation, par exemple :

 Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:52.0) Gecko/20100101 Firefox/52.0.

Quel est l’User\_agent de notre robot ?

Essayez la commande suivante:  requests.utils.default\_user\_agent().

En l’exécutant vous allez apprendre que votre robot s’appelle ‘python-requests/2.21.0’. Il ne figure pas sur la liste des mauvais robots, mais cela ne signifie pas pour autant qu’il est le bienvenu. Le serveur de seloger.com détecte rapidement son comportement ‘robotique’ et le bloque temporairement.

À la différence des humains, les robots :

* N’utilisent pas les navigateurs et possèdent donc un mauvais User\_agent
* Restent très peu de temps sur une page
* Demandent plusieurs pages à la fois depuis la même adresse IP

Afin d’éviter d’être bloqué, on peut demander à notre robot de se présenter correctement, comme un humain, en précisant un bon User\_agent.

Pour ne pas endommager le site et se comporter de façon plus “humaine”, on peut également demander à notre robot de patienter quelques secondes sur une page avant de passer à une autre. Enfin, à chaque passage, nous pouvons changer l’adresse IP du robot.

Voici comment réaliser ces ajustements. Commençons par créer un pool de proxies. On trouve plusieurs listes de proxies gratuits sur Internet.

Pour créer notre pool, nous avons utilisé celui-ci: [www.proxy-list.download/HTTPS](http://www.proxy-list.download/HTTPS).

Il faudra la télécharger et l’importer dans notre espace de travail.

On crée ensuite un [itérateur](https://docs.python.org/3.5/library/itertools.html#itertools.cycle), un objet qui nous permettra changer de proxy à chaque connexion à l’aide de la fonction next().

import itertools as itproxies = pd.read\_csv('proxy\_list.txt', header = None)  
proxies = proxies.values.tolist()  
proxies = list(it.chain.from\_iterable(proxies))  
proxy\_pool = it.cycle(proxies)  
proxy = next(proxy\_pool)

Maintenant, apprenons notre robot à se présenter correctement et à être moins impatient. Pour le faire, nous utilisons les librairies time,random et fake-useragent:

import random  
import time  
# !pip install fake-useragent  
from fake\_useragent import UserAgentua = UserAgent()  
time.sleep(random.randrange(1,5))

Par conséquent, on modifie notre fonction requests.get():

response = requests.get(i,proxies={"http": proxy, "https": proxy}, headers={'User-Agent': ua.random},timeout=5)

Elle contient maintenant unproxy qui change à chaque nouvelle exécution de la ligne proxy = next(proxy\_pool) et une fausse identité ua.random. Nous avons ajouté également l’argument timeout=5 qui signifie que chaque tentative de connexion à une page dure 5 secondes maximum.

Les proxies gratuits ne sont jamais très fiables. Afin de revenir sur une page que notre fonction requests.get() n’est pas parvenue à ouvrir à cause d’une mauvaise connexion, nous avons ajouté lacondition suivante :

while len(pages) > 0:  
 try:  
 ...  
 pages.remove(i)  
 except:  
 print("Skipping. Connnection error")

La boucle while répétera une tentative de connexion jusqu’à l’épuisement des pages à consulter dans notre liste.

Avec une telle requête, le serveur ne saura jamais que vous êtes un robot et vous obtiendriez toutes les informations souhaitées.

Notons toutefois que seloger.com interdit le scraping dans [ses conditions d’utilisation](https://www.seloger.com/CGU_politique_de_confidentialite.html).

L’exécution du code de ce tutoriel relève donc entièrement de votre responsabilité !

**Récapitulation**

Les techniques de scraping facilitent la collecte et le traitement des données dispersées sur Internet. Pour réaliser un scraping il faut :

* Créer une liste des pages web à parcourir.
* Localiser l’information qui vous intéresse dans le code source de ces pages. Avoir des bases en HTML vous facilitera cette étape du travail.
* Une fois que vous avez identifié les balises à extraire, vous pouvez créer une boucle qui répète la même opération pour plusieurs pages.
* Si le site vous bloque, changez d’User\_agent et utilisez un proxy.

La structure des pages web changent constamment. Il suffit d’une légère modification pour rendre votre code obsolète. Il faudra donc de temps en temps mettre à jour votre scraper.

**Les sites n’aiment pas les scrapers. La collecte de leurs données peut être considérée comme une violation de la propriété intellectuelle ou encore comme un vol des données personnelles. Lisez donc attentivement les conditions d’utilisation des sites que vous voulez scraper.**

|  |  |
| --- | --- |
| import pandas as pd | |
|  | import time |
|  | import bs4 |
|  | import random |
|  | import requests |
|  | # !pip install fake-useragent |
|  | from fake\_useragent import UserAgent |
|  | import itertools as it |
|  |  |
|  | token = 'https://www.seloger.com/immobilier/locations/immo-paris-75/bien-appartement/?LISTING-LISTpg=' |
|  |  |
|  | def get\_pages(token, nb): |
|  | pages = [] |
|  | for i in range(1,nb+1): |
|  | j = token + str(i) |
|  | pages.append(j) |
|  | return pages |
|  |  |
|  | pages = get\_pages(token,295) |
|  |  |
|  | # https://www.proxy-list.download/HTTPS |
|  | proxies = pd.read\_csv('proxy\_list.txt', header = None) |
|  | proxies = proxies.values.tolist() |
|  | proxies = list(it.chain.from\_iterable(proxies)) |
|  |  |
|  | def get\_data(pages,proxies): |
|  |  |
|  | df = pd.DataFrame() |
|  | parameters = ['data-prix','data-codepostal','data-idagence','data-idannonce','data-nb\_chambres','data-nb\_pieces','data-surface','data-typebien'] |
|  | ua = UserAgent() |
|  | proxy\_pool = it.cycle(proxies) |
|  |  |
|  | while len(pages) > 0: |
|  | for i in pages: |
|  | # on lit les pages une par une et on initialise une table vide pour ranger les données d'une page |
|  | df\_f = pd.DataFrame() |
|  | # itération dans un liste de proxies |
|  | proxy = next(proxy\_pool) |
|  | # essai d'ouverture d'une page |
|  | try: |
|  | response = requests.get(i,proxies={"http": proxy, "https": proxy}, headers={'User-Agent': ua.random},timeout=5) |
|  | time.sleep(random.randrange(1,5)) |
|  | # lecture du code html et la recherche des balises <em> |
|  | soup = bs4.BeautifulSoup(response.text, 'html.parser') |
|  | em\_box = soup.find\_all("em", {"class":"agency-website"}) |
|  | # extraction des données |
|  | for par in parameters: |
|  | l = [] |
|  | for el in em\_box: |
|  | j = el[par] |
|  | l.append(j) |
|  | l = pd.DataFrame(l, columns = [par]) |
|  | df\_f = pd.concat([df\_f,l], axis = 1) |
|  | df = df.append(df\_f, ignore\_index=True) |
|  | pages.remove(i) |
|  | print(df.shape) |
|  | except: |
|  | print("Skipping. Connnection error") |
|  |  |
|  | return df |
|  |  |
|  | data = get\_data(pages,proxies) |